Computer Architecture Project
Pipelined Harvard Processor
Spring 2021

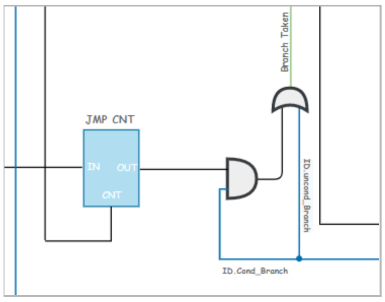
Team 14

Moaaz Ashraf Zaki	1170353
Mohammed Mohammed Saad	1170044
Mariam Mohamed El Baz	1170546
Fatema Fawzy Mohamed	1170530

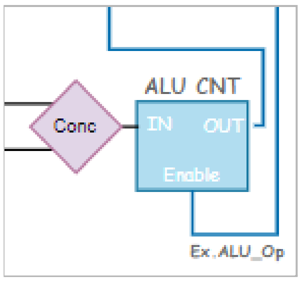
Design Changes:

- The sizes of the buffers between stages are extended because more information was needed to be passed along the pipe.

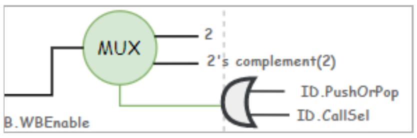
- In the branching instructions, the Branch_Taken flag is taken as the output of the JumpControl register as all the checks are done internally. Hence, the and and or gates are removed. The blue signal was taken from the control unit in the initial design, but now there is no need for it as the operation bits are taken as input to the JumpControl register and checks are done based on them.



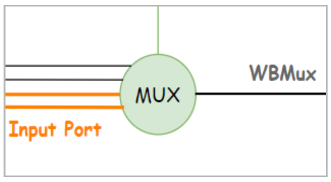
- Using the same previous concept, the ALU CNT is removed and its inputs are now inputs to the ALU itself and all the checks are done internally as well.



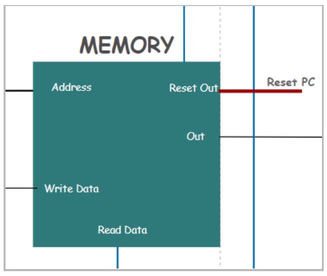
- For the MUX used for the SP to choose whether to add or subtract 2 from the current SP, the OR gate is removed and 1 selecting signal is used to decide whether to add or subtract according to the instruction.



- The input port is now taken from the fetch stage, not the write back.



- The Reset PC is moved from the main memory (RAM) to the instructions memory (ROM).



Pipelining Hazards:

1) Structural Hazards:

- **Problem:** Using the same memory in the fetch stage (for fetching the instructions) and memory stage (for data access.)

  Solution: Duplicate the memory resource.

- **Problem:** Using the same register in the decode stage (for reading from register) and write back stage (for writing to register.)

  Solution: Write back stage will be on the negative edge(first half cycle), and decode stage will be on the positive edge(second half cycle.)

2) Data Hazards:

- **Problem:** RAW data dependency

  Solution: Full forwarding. (ALU-ALU) and (Memory-ALU)

- **Problem:** LOAD USE scenario

  Solution: Stall for 1 clock cycle till the data is ready in the memory stage then forward normally from memory to ALU.

3) Control Hazards:

- **Problem:** An instruction changes the PC.

  Solution: Static branch prediction (Always predict not taken.)

- **Problem:** Reading an outdated value of the register in the decode stage.

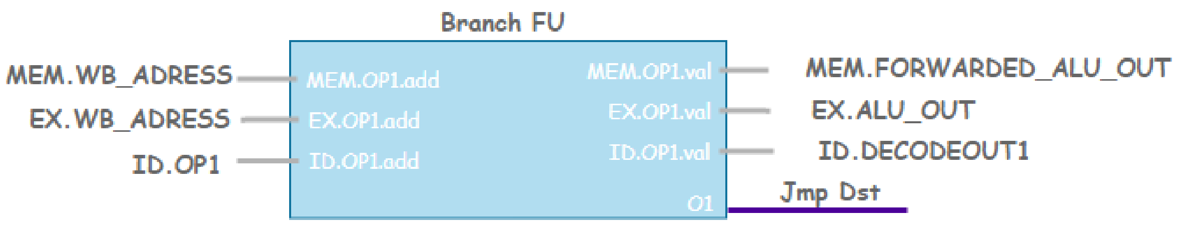
  Solution: Branch forwarding unit to forward the correct value of the registers used in the branching instruction.

<u>Notes about the branch instructions:</u>

We implemented the branch instructions but without handling all of its hazards.

As we were taking the branch decision in the decode stage, and we only implemented ALU-ALU and MEM-ALU forwarding, sometimes the jump goes to an old value stored in the register as it did not have a chance to write back yet.

As a solution to this, we implemented another forwarding unit called “BranchForwardingUnit”, its job is similar to the normal forwarding unit but it forwards to the decode stage instead of the execute stage. Its schematic is shown down below.



The JMP DST output is now the correct updated/forwarded value of the register which we are going to use its location to jump to.

Unfortunately, time did not permit us to integrate this module with the main pipeline.

<u>General notes and assumptions:</u>

1) The flags are changed in the execute stage.
2) The output port value is updated in the execute stage.

Notes on how to run the code:

How to run existing test cases:

There is a folder called tests in the root directory of the project. Inside of it you will find:

1) 1Operand.do: file used to test the 1 operand test case.
2) 2Operands.do: file used to test the 2 operands test case.
3) Memory.do: file used to test the memory test case.

If you would like to create a new test case:

1) There is a file called assemblyConverter.py inside the scripts folder found in the main directory.
2) Run this file using any python cmd (like anaconda prompt)
3) It will ask you for the name of the input file (that should be the .asm file.) Make sure to enter only the name of the file without its extension.
4) It will ask you to enter the name of the output file. Enter only the name of the file without any extensions and it will produce a .txt file with the binary memory contents.
5) Copy the memory contents of the produced file to the file named rom_content_bin.txt in the instructions folder found in the root directory.
6) Make a do file and run it.